第三章
Java 與 Microsoft Office

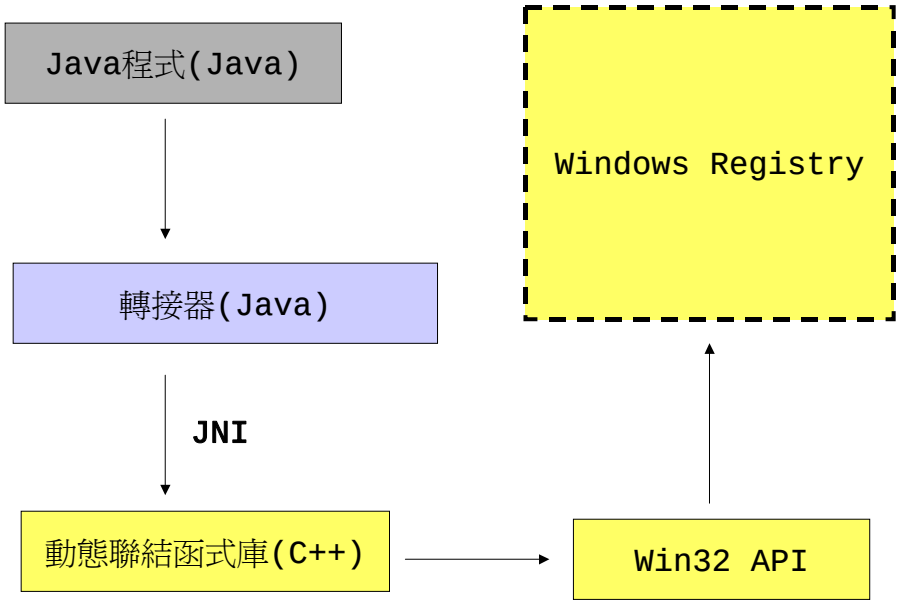
如果你相信了他的經驗，那麼你的極限就會侷限在他的經驗之中。

簡介

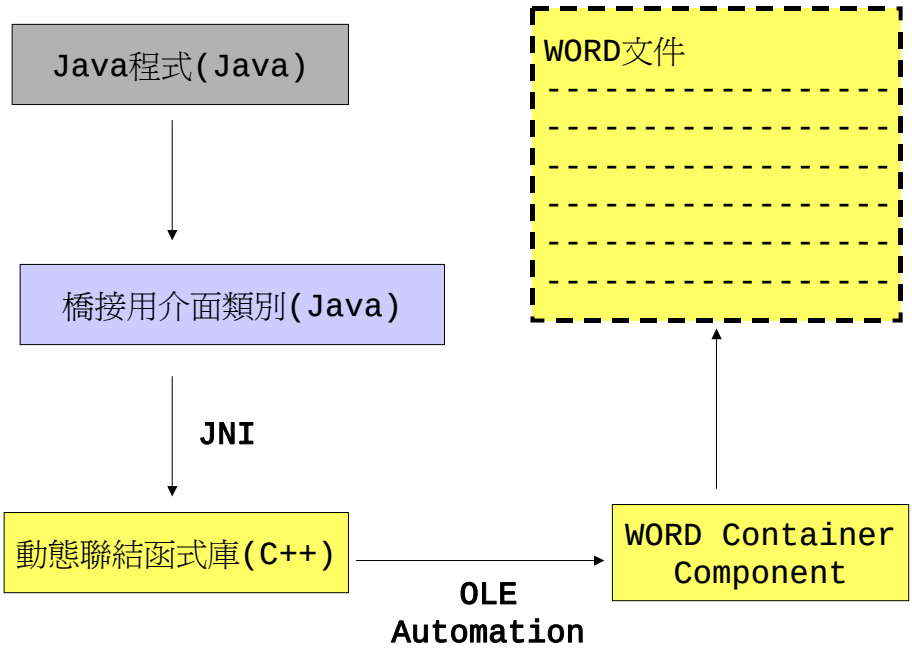
記得不久前，在網路上看到一位網友寫著:「Java 把程式設計師和底層作業系統隔開，因而讓不同平台上的程式可以用同一種方式開發。但是也因為如此，Java 讓程式設計師和底層作業系統之間變得毫無關聯，使得很多原本用 Visual Basic、Delphi 等原生開發工具很容易做到的事情，在 Java 中卻難以做到。」其實，就基本的概念來說，這句話是正確的，也說明了 Java 在某些功能上的確力不從心。可是，這不就是 Java 之所以為 Java 的理由嗎?

對實際參與 Java 應用程式開發的工程師來說，往往因為某些特殊因素(通常是客戶和老闆的要求)，需要完成一些與作業系統結合較緊密的程式，比方說要從 Java 應用程式裡頭去操作 Windows Registry、或是要讓 Java 應用程式可以操作未公開格式的 Microsoft WORD 文件檔。這些功能都是標準的 Java 類別函式庫沒有提供的，所以，除了到網路上搜尋是否有廠商提供類似功能的元件之外，工程師幾乎一籌莫展。這個時候，JNI(Java Native Interface)就是最好的萬靈丹。

如果要從 Java 應用程式裡頭去操作 Windows Registry，那麼我們只要寫個簡單的轉接器(adapter)，讓 Java 叫用那些被包裝好、用來操控 Windows Registry 的 Win32 API 即可，如下圖所示:



然而，如果要讓 Java 應用程式可以操作未公開格式的 Microsoft WORD 文件檔，可就沒有那麼簡單了，在無法得知檔案格式的情況下，我們該如何解決呢? 於是，問題回到基本面:「既然只有 WORD 才能讀取 WORD 文件檔，那我們就請 WORD 幫我們讀不就好了嗎? 」，此時，就必須要動用到 OLE Automation 技術，方法如下圖所示:



本文所要討論的重點，就是如何在 Java 應用程式之中使用 Microsoft Office 之中所提供的軟體元件。本文的程式碼純粹為了便於展示用，沒有做過任何最佳化。如果您要讓程式碼變得更有擴充性，請使用適當的 Design Pattern

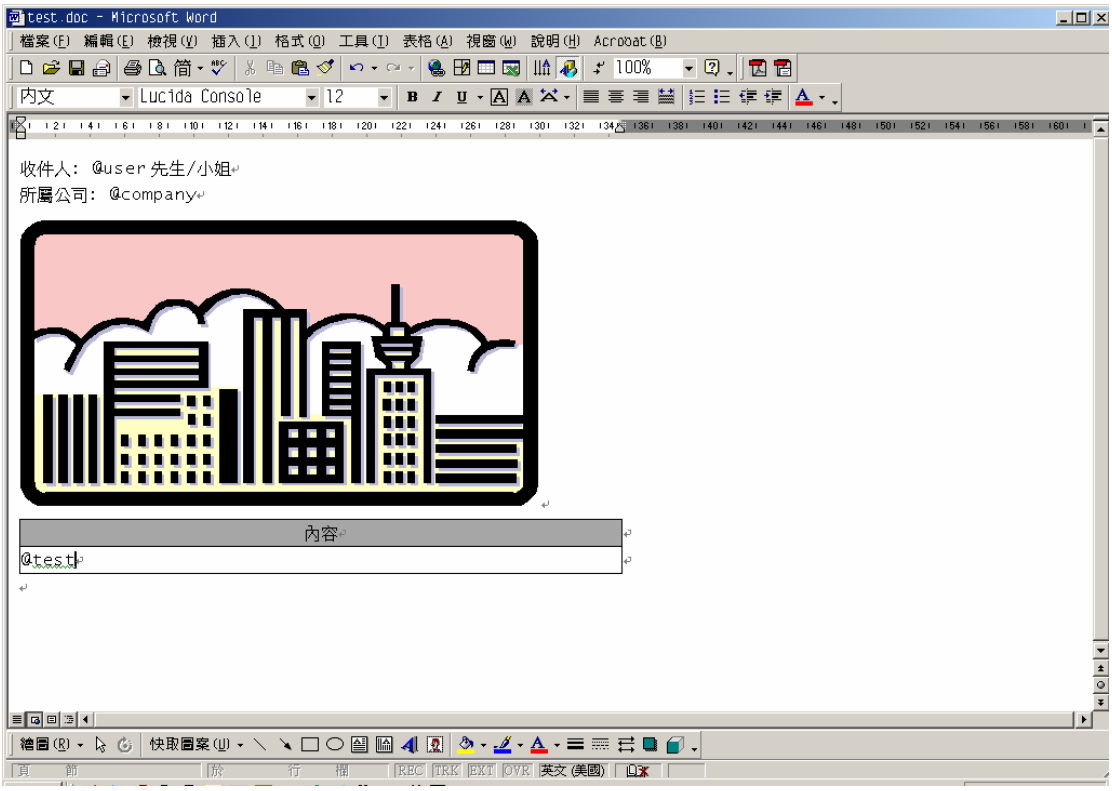
即可，程式基本的運作原理是不變的。

本章目的

本章將教大家從無到有製作一個「套表列印模組」。此模組的功能說明如下:

功能緣起:

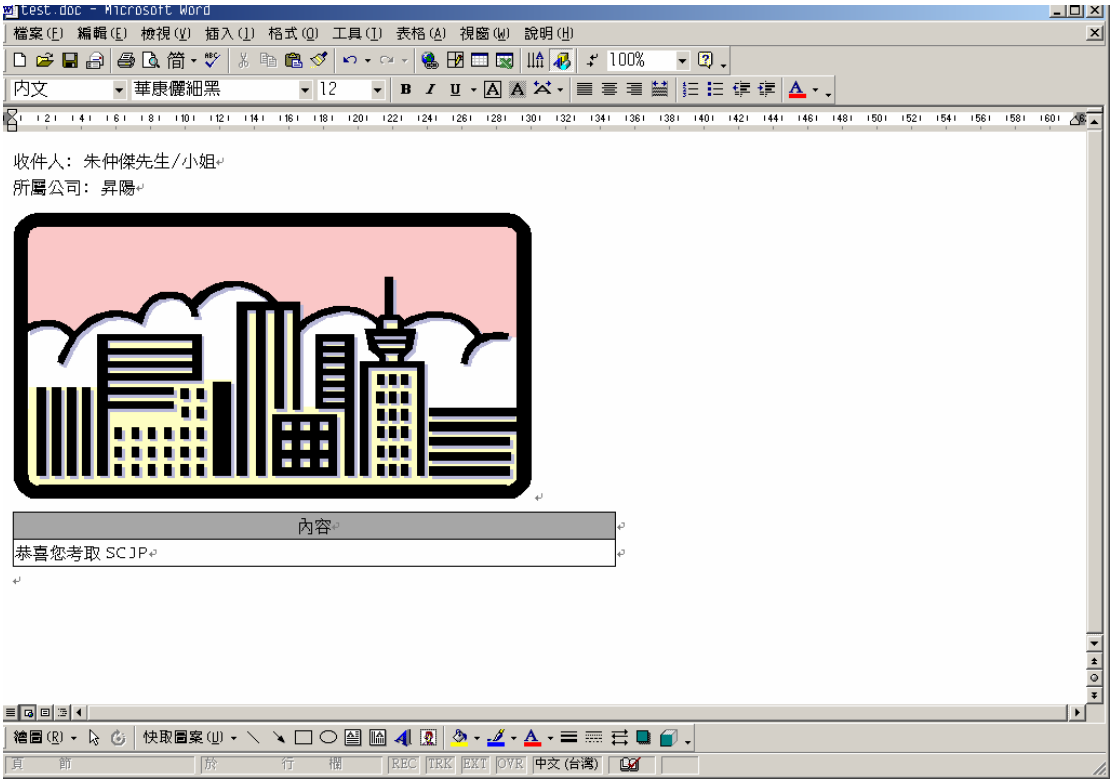
在一般公司行號的辦公室當中，幾乎是以 Microsoft Office 作為文件處理之首，而 Microsoft WORD 更可以製作出圖文並茂的文件，這並非使用既有的列印元件(如:Borland C++Builder 提供的 Quick Report)就可以簡單達到的事。因此，我們希望先用 Microsoft WORD 製作出圖文並茂的文件樣版，如下圖所示:



然後，我們只要在 Java 程式之中提供一個字串陣列:

String context[] = { "test:恭喜您考取 SCJP" , "company:昇陽" , "user:朱仲傑" } ;

當 Java 程式叫用了我們所撰寫的套表列印模組之後，樣版就會自動將出現『@test』的地方取代成「恭喜你考取 SCJP」、『@company』的地方取代成「昇陽」、『@user』的地方取代成「朱仲傑」，結果應如下圖所示:



當然，當您的本文中真的需要使用"@"和":"時，這樣的設計會造成若干衝突，此時可能要動用到跳脫字元(escape character)的設計，請讀者自行衡量。

在本文中，筆者將使用 C++製作一個可以呼叫 OLE Automation 的動態聯結函式庫，並從 Java 應用程式之中利用 JNI 技術呼叫此動態聯結函式庫，以達到前述我們所期望的功能。

基本技能

要了解本文之中的所有程式碼，您必須具備下列幾項技能:

1. Java 程式的撰寫能力。
2. C++程式的撰寫能力。
3. Visual Basic for Application(VBA)程式的撰寫能力。

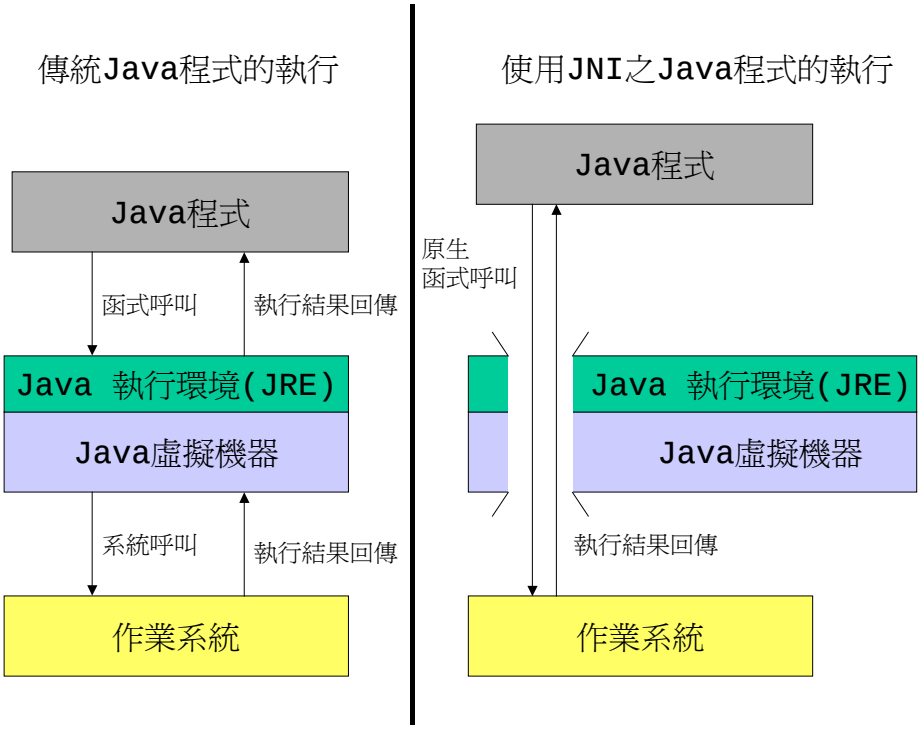
不過有鑒於一般的程式設計師通常只將自己的焦點放在其中一項技能，所以本文並不打算假設閱讀本文的讀者對這三項技能都非常熟悉。我的一位醫師好友邱建勳曾經提醒我，好的老師一定要教學生拿釣竿，讓他們自己具備捕魚的能力，所以本文將會告訴大家，如果以後設計一個類似的功能模組時，要怎麼樣才能順利完成。

即使如此，本文所用到的程式碼仍有其進入門檻。如果有必要，請多參考本文所提到的所有相關資源。

架構

在整個範例程式之中，最重要的技術莫過於 JNI(Java Native Interface)，

這是 Java 程式與 C++程式之所以可以協同運作的重要關鍵，所以我們必須了解 JNI 究竟是一種什麼樣的技術。



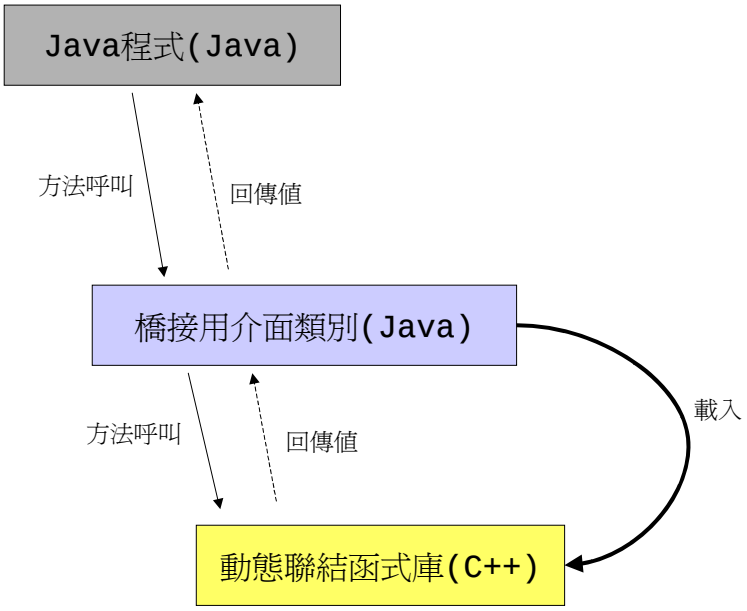
從上圖中，讀者可以發現，Java 程式都因為 Java 虛擬機器的關係，和底層完全地隔離開來。可以一旦使用 JNI 之後，Java 虛擬機器會開放一扇門戶，和 Java 程式可以突破 Java 虛擬機器的限制，直接呼叫底層作業系統的原生函式庫。

但是這麼一來，會引發安全性的問題。因為 Java 的安全機制完全由 Java 執行環境之中的 Security Manager 來控制，並配合 Policy File 來設定控制權限。因此，如果要使用 JNI，一定要先考量安全性的議題，否則您的 Java 程式就在不知不覺中為破壞者開啟了後門而不自覺。您必須在 Java 層級(也就是執行權限尚未穿越 Java 虛擬機器前)就完成安全性上的限制。關於此問題，請參考 Java Security 相關書籍。

接下來我們就要開始整個功能模組的建構程序。

功能模組建構程序一：設計橋接用的介面類別

首先，我們必須製作一個可以供 Java 程式呼叫用的 Java 類別，這個類別的主要功能是作為橋接之用，除了作為 Java 與 C++之間的介面之外，也用來幫助 Java 執行環境載入所需要的動態聯結函式庫，如下圖所示:



虛線部份表示:『來自於 Java 應用程式的方法呼叫，不一定需要有傳回值回傳(void) 』。

橋接介面類別的原始碼如下:

JavatoWord.java

```
/*************************************************************
Java2Word.java
JNI 橋接介面物件
2002 Jan by 王森
moli@pchome.com.tw
*************************************************************/
package com.sun.edu ;
public class JavatoWord {
	/*
	 原生函式
	 context ==> 所有文字內容所構成的陣列
	      count = context.length
	*/
	public native void NativeToWord(String[] context,int count) ;
	/*
	在使用原生函式之前，再次確認參數的正確性，儘量讓錯誤的處理在 Java
	之中完成。
	*/
	public boolean ToWord(String[] context)
```

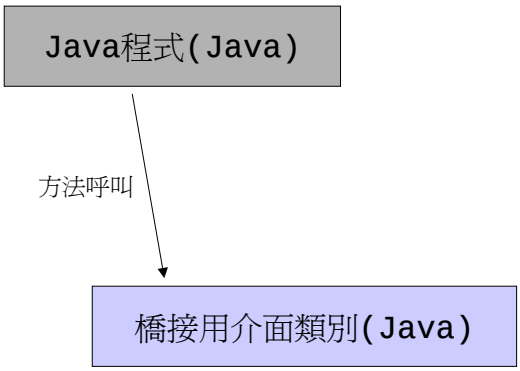
```
    {
        try
        {
            NativeToWord(context,context.length) ;
            return true ;
        }catch(Exception e)
        {
            return false ;
        }
    }

    static
    {
        //我們把實做轉換至 WORD 的函式的模組做成 DLL(動態連結函式庫)檔,
        //取名叫 Transfer.dll,所以在這裡要載入此 DLL
        System.loadLibrary("Transfer") ;
    }
}
```

此類別有兩個特殊的地方:

1. NativeToWord 方法前面加上了 native 修飾字。
   此修飾字的意思即是告訴編譯器:"這個函式的實際內容並非以 Java 所撰寫，而是以其他語言所撰寫"。
2. 使用了靜態初始化區塊。
   當 JavatoWord.class 被類別載入器(class loader)載入時，靜態初始化區塊就會被呼叫，並載入名為 Transfer.dll 的動態聯結函式庫。請注意，靜態初始化區塊在整個 Java 程式的生命週期之中，只有在被類別載入器載入的時候才被呼叫一次，往後不會再被叫用。

請注意，JavatoWord 這個類別屬於 com.sun.edu 這個 package，所以一定要放置在相對路徑 com\sun\edu\底下，否則編譯的時候雖然不會出錯，執行的時候會出現執行時期例外。相關內容請參考本書後面討論 package 機制的章節，這裡不再贅述。

功能模組建構程序二：撰寫使用橋接介面類別的 Java 程式



test.java

```
/**********************************************************
test.java
JNI 測試物件
2002 Jan by 王森
moli@pchome.com.tw
**********************************************************/
import com.sun.edu.* ;
class test
{

	//主程式開始
	public static void main(String args[])
	{
		//取得 JNI 橋接介面物件
		JavatoWord my = new JavatoWord() ;

		String context[] =
		{ "test:恭喜您考取 SCJP" , "company:昇陽" ,  "user:朱仲傑" } ;

		//開始進行轉換工作
		if(my.ToWord(context))
		{
```

```
            System.out.println("呼叫成功") ;
        }else
        {
            System.out.println("傳入陣列有誤,可能是元素數量不正確") ;
        }
    }//end of main
}//end of class
```

■功能模組建構程序三：產生編譯動態聯結函式庫時所需要的 C/C++引入檔

請使用指令:

javah -classpath . com.sun.edu.JavatoWord

```
E:\Java2Word>javah -classpath . com.sun.edu.JavatoWord

E:\Java2Word>dir
 磁碟區 E 中的磁碟是 INTERNAL
 磁碟區序號:  2058-927C

 目錄:  E:\Java2Word

2001/12/27  10:34a      <DIR>          .
2001/12/27  10:34a      <DIR>          ..
2001/07/16  07:54a                 463 readme.txt
2000/07/30  03:52p                  73 test.bat
2000/08/03  11:07p              19,456 test.doc
2001/12/27  10:39a                 702 test.java
2001/12/27  10:39a                 693 test.class
2001/12/27  10:34a      <DIR>          bat
2001/12/27  10:34a      <DIR>          com
2001/12/27  10:34a      <DIR>          dll
2001/12/27  10:45a                 503 com_sun_edu_JavatoWord.h
               6 個檔案         21,890 位元組
               5 個目錄   1,316,352,000 位元組可用

E:\Java2Word>
```

就可以在目錄之中發現 javah.exe 幫我們在目錄下產生了一個叫做 com_sun_edu_JavatoWord.h 的 C++引入檔，其內容如下:

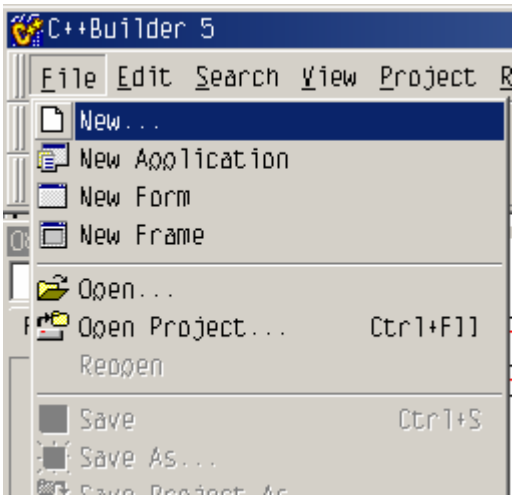
com_sun_edu_JavatoWord.h

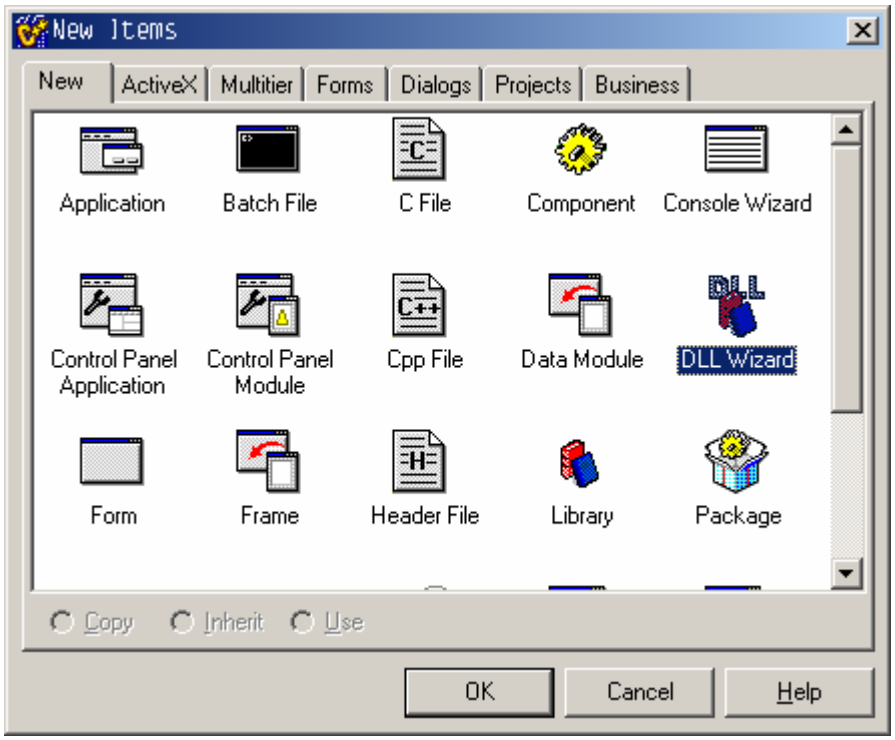
```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class com_sun_edu_JavatoWord */
#ifndef _Included_com_sun_edu_JavatoWord
```

```
#define _Included_com_sun_edu_JavatoWord
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:       com_sun_edu_JavatoWord
 * Method:      NativeToWord
 * Signature: ([Ljava/lang/String;I)V
 */
JNIEXPORT void JNICALL Java_com_sun_edu_JavatoWord_NativeToWord
   (JNIEnv *, jobject, jobjectArray, jint);

#ifdef __cplusplus
}
#endif
#endif
```

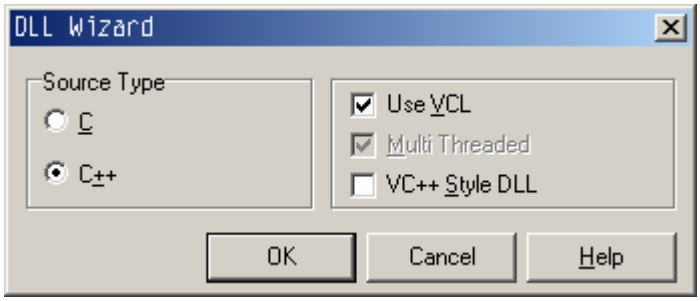
功能模組建構程序四：建立套表列印模組的基本架構

請先開啟您的 Borland C++Builder，並開啟一個新專案，如下圖:





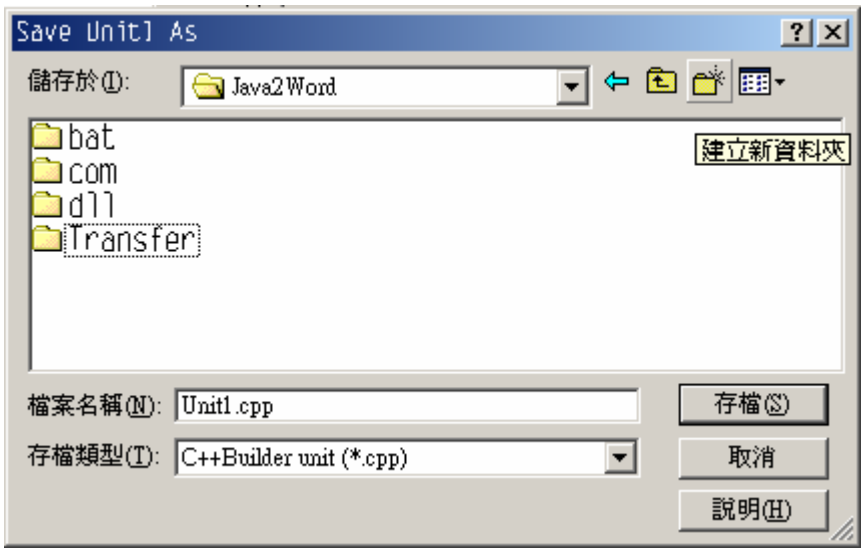
請在選擇 DLL Wizard 之後，按下 OK 鈕，接著會出現底下畫面。



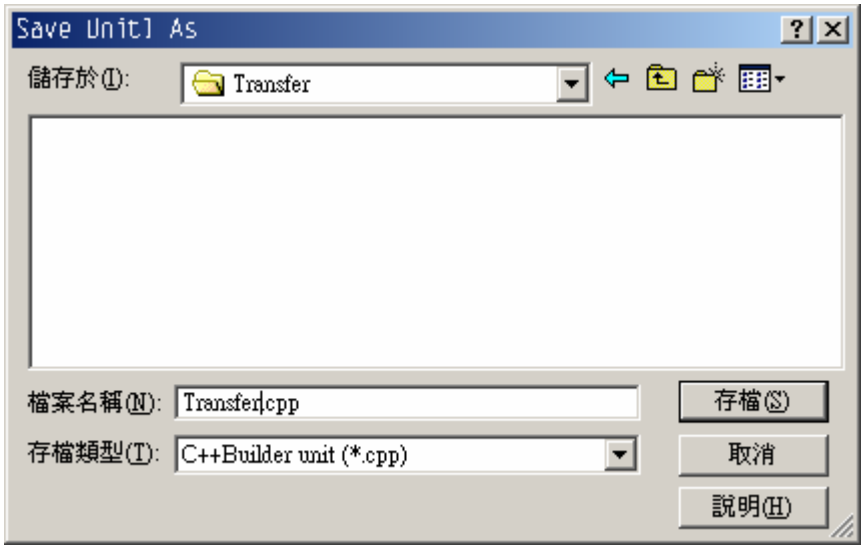
請直接按下 OK 鈕即可。精靈會自動幫我們建構專案，並產生對應的動態聯結函式庫程式碼主幹。

接著，請按下 Save All 鈕，如下圖:

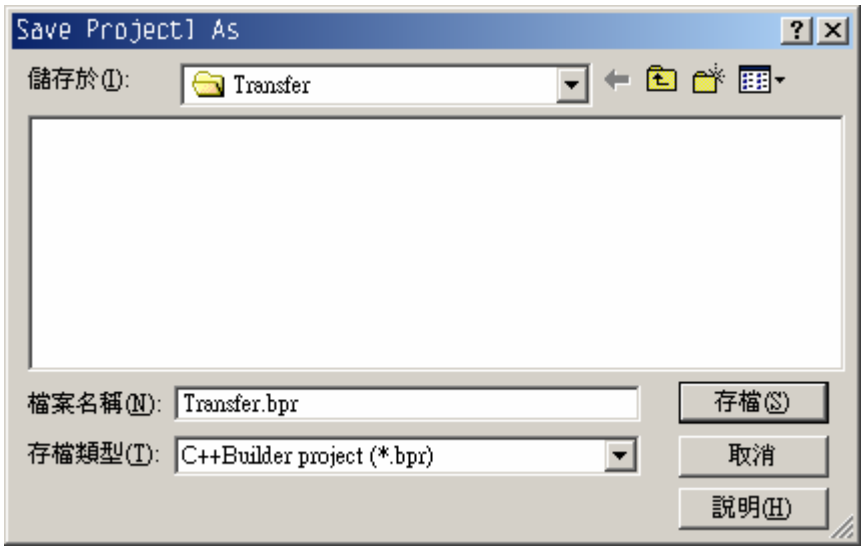
請先建立一個目錄，名為 Transfer，如下圖:



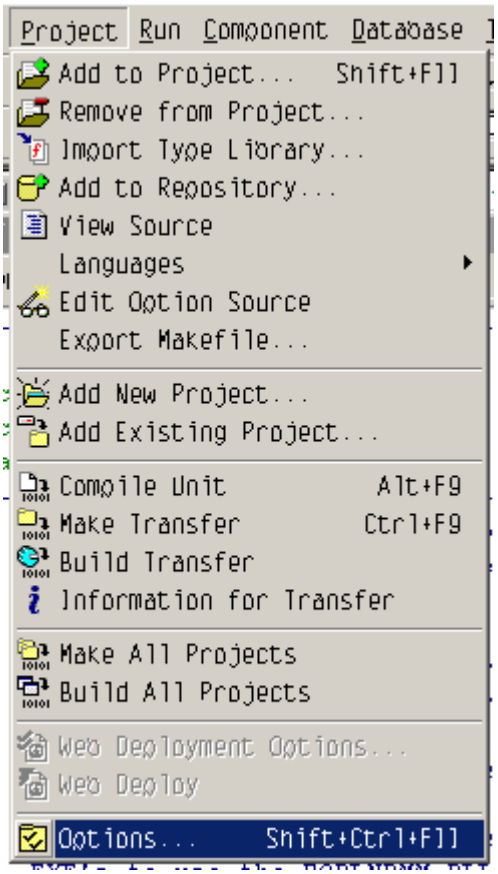
進入 Transfer 目錄，將主程式碼檔名儲存為 Transfer.cpp，如下圖:



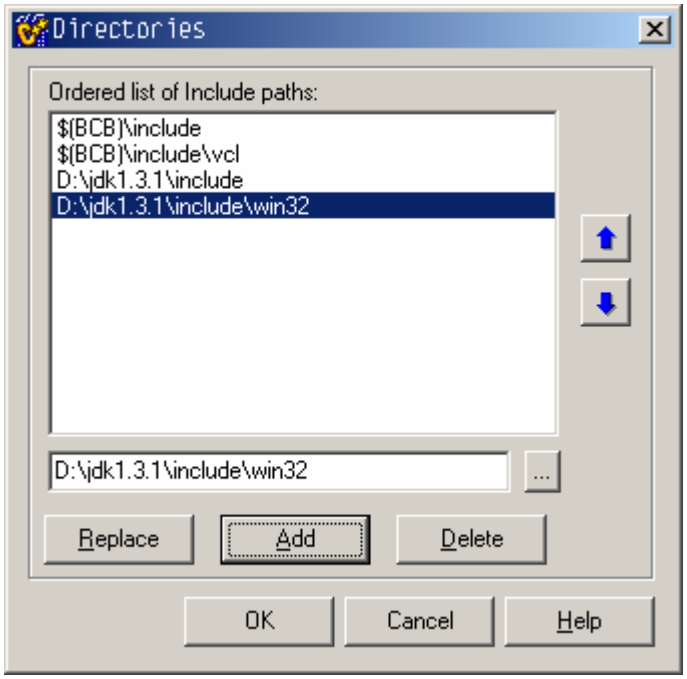
同時也把專案檔檔名改儲存為 Transfer.bpr，如下圖:



接著，請選擇 Project | Options 選項，我們要設定一些環境變數:



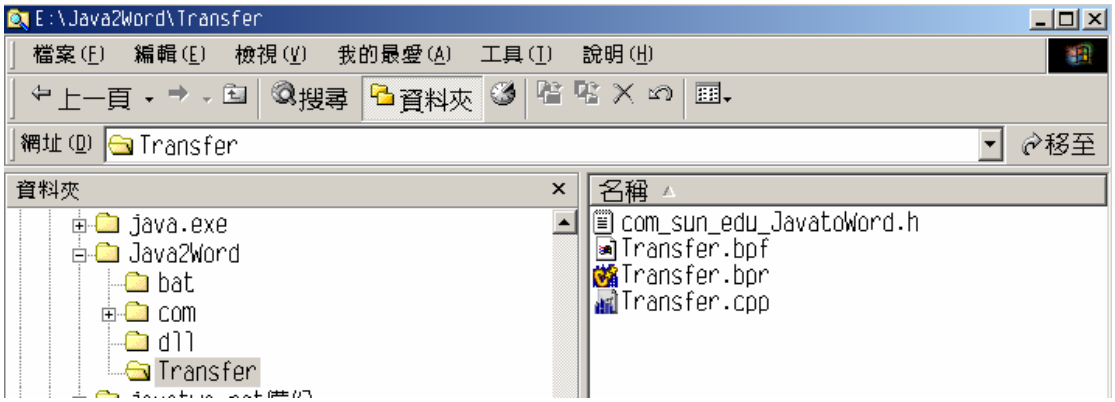
進入 Project Options 對話方塊之後，請先選擇 Directories and Conditionals 次頁，將兩個使用 JNI 時一定要使用的引入檔之所在目錄加進來，他們分別是 <jdk 安裝目錄>\include 與<jdk 安裝目錄>\include\win32，如下圖:



(注意: 筆者將 JDK 安裝在 d:\jdk1.3.1，請各位讀者依自己電腦中 JDK 所在位

置作調整。)

接著，請將 com_sun_edu_JavatoWord.h 放到專案所在的目錄下，所有與該專案相關的檔案如下圖所示:



完成上述步驟之後，請將 Transfer.cpp 修改成底下的樣子:

Transfer.cpp

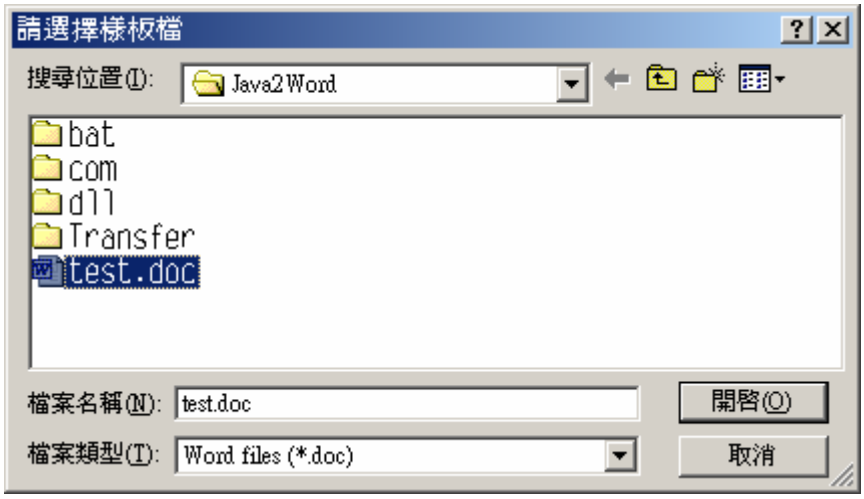
```
#include <vcl.h>
#include <windows.h>
#include "com_sun_edu_JavatoWord.h"
#pragma hdrstop

#pragma argsused
int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason, void* lpReserved)
{
        return 1;
}

JNIEXPORT void JNICALL Java_com_sun_edu_JavatoWord_NativeToWord
  (JNIEnv *env, jobject obj, jobjectArray context, jint count)
{
        //讀入樣板檔
        TOpenDialog* templatefile = new TOpenDialog(NULL) ;
        templatefile->Title = "請選擇樣板檔" ;
        templatefile->Filter = "Word files (*.doc)|*.doc" ;
        AnsiString filename ;
        if(templatefile->Execute())
        {
                filename = templatefile->FileName ;
        }
```

```
        //檢查樣板檔是否存在
        if(!FileExists(filename))
        {
                ShowMessage("您所輸入的檔案並不存在,請重新輸入") ;
                delete templatefile ;
                return ;
        }
        delete templatefile ;
        ShowMessage("您選擇的檔名為" + filename) ;
}
```

完成後，請使用選單上的 Project | Make Transfer 建立動態聯結函式庫(.dll 檔)。然後到 DOS 視窗，執行指令:

java –cp . –Djava.library.path=.\Transfer test

如果執行正常，螢幕輸出應該如下圖:



選擇了某個檔案之後，按下開啟之後，螢幕上會出現:

DOS 視窗的輸出為:

```
E:\Java2Word>java -cp . -Djava.library.path=.\Transfer test
呼叫成功

E:\Java2Word>
```

接下來，我們要測試從 Java ➔ C++時的參數傳遞是否正確，因此我們把程式碼改成底下的樣子:

Transfer.cpp

```
#include <vcl.h>
#include <windows.h>
#include <iostream.h>
#include "com_sun_edu_JavatoWord.h"
#pragma hdrstop

#pragma argsused
int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason, void* lpReserved)
{
        return 1;
}

JNIEXPORT void JNICALL Java_com_sun_edu_JavatoWord_NativeToWord
  (JNIEnv *env, jobject obj, jobjectArray context, jint count)
{
        //讀入樣板檔
        TOpenDialog* templatefile = new TOpenDialog(NULL) ;
        templatefile->Title = "請選擇樣板檔" ;
        templatefile->Filter = "Word files (*.doc)|*.doc" ;
        AnsiString filename ;
        if(templatefile->Execute())
        {
                filename = templatefile->FileName ;
        }
        //檢查樣板檔是否存在
        if(!FileExists(filename))
        {
                ShowMessage("您所輸入的檔案並不存在,請重新輸入") ;
                delete templatefile ;
                return ;
        }
        delete templatefile ;
        ShowMessage("您選擇的檔名為" + filename) ;

        cout << "接收到 " << count << " 個字串" << endl;
        for(int i = 0 ; i < count ; i++)
        {
```

```
                    //取出 context
                    jstring jsdata =
                    (jstring)(*env).GetObjectArrayElement(context,i) ;
                    wchar_t* csdata =
                    (wchar_t*)(*env).GetStringChars(jsdata,NULL) ;
                    int size = (*env).GetStringLength(jsdata) ;
                    WideString wtmp = WideString(csdata,size) ;
                    ShowMessage(wtmp) ;
        }
}
```

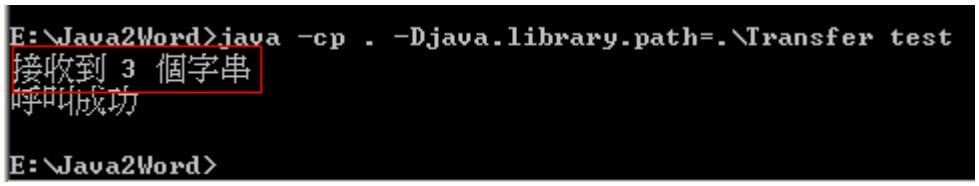
完成後，請使用選單上的 Project | Make Transfer 重新建立動態聯結函式庫(.dll 檔)。然後到 DOS 視窗，執行指令:

java –cp . –Djava.library.path=.\Transfer test

如果執行正常，除了輸出選擇的檔案名稱之外，螢幕還會額外輸出:

等其他兩個視窗(因為我們傳遞了三個參數)。

DOS 視窗的輸出如下:



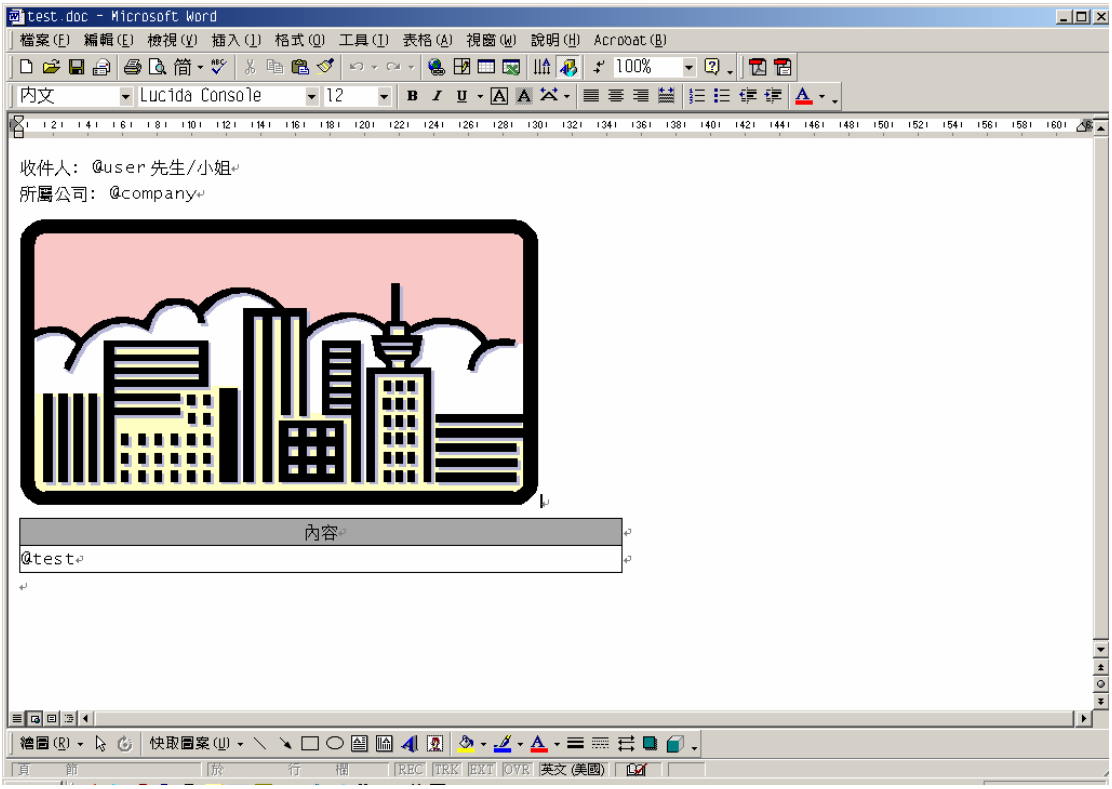
這麼一來表示從 Java ➔ C++之間的參數傳遞沒有任何問題。

功能模組建構程序五: Visual Basic for Application

從 Java ➔ C++之間的參數傳遞沒有問題了，接著我們要使用 C++來操控 VBA(Visual Basic for Application，也就是一般 Microsoft Office 術語之中常說的"巨集"指令)，所使用的技術就是 OLE Automation。可是，VBA 指令怎麼下呢? 筆者所使用的方法步驟如下:

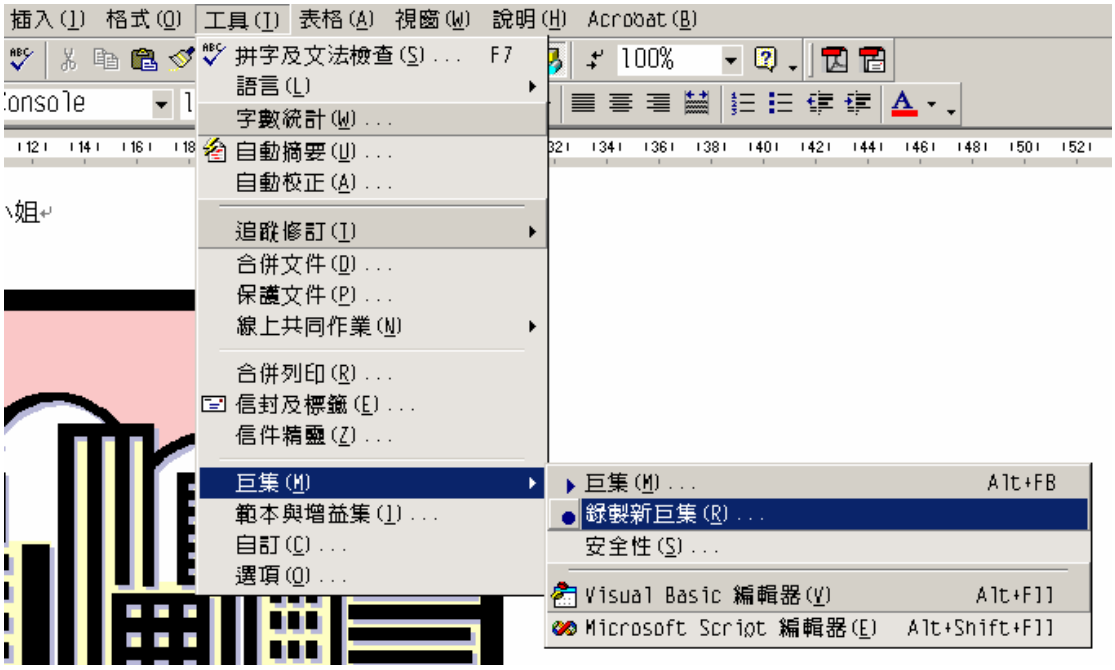
1.製作樣版檔

首先，先撰寫我們所需要的 WORD 文件樣版檔如下:



2.錄製巨集

請選擇 Microsoft WORD 選單上的 『工具 | 巨集 | 錄製新巨集』，如下圖所示:



在錄製巨集對話方塊中填入 test 為巨集名稱，如下圖所示:

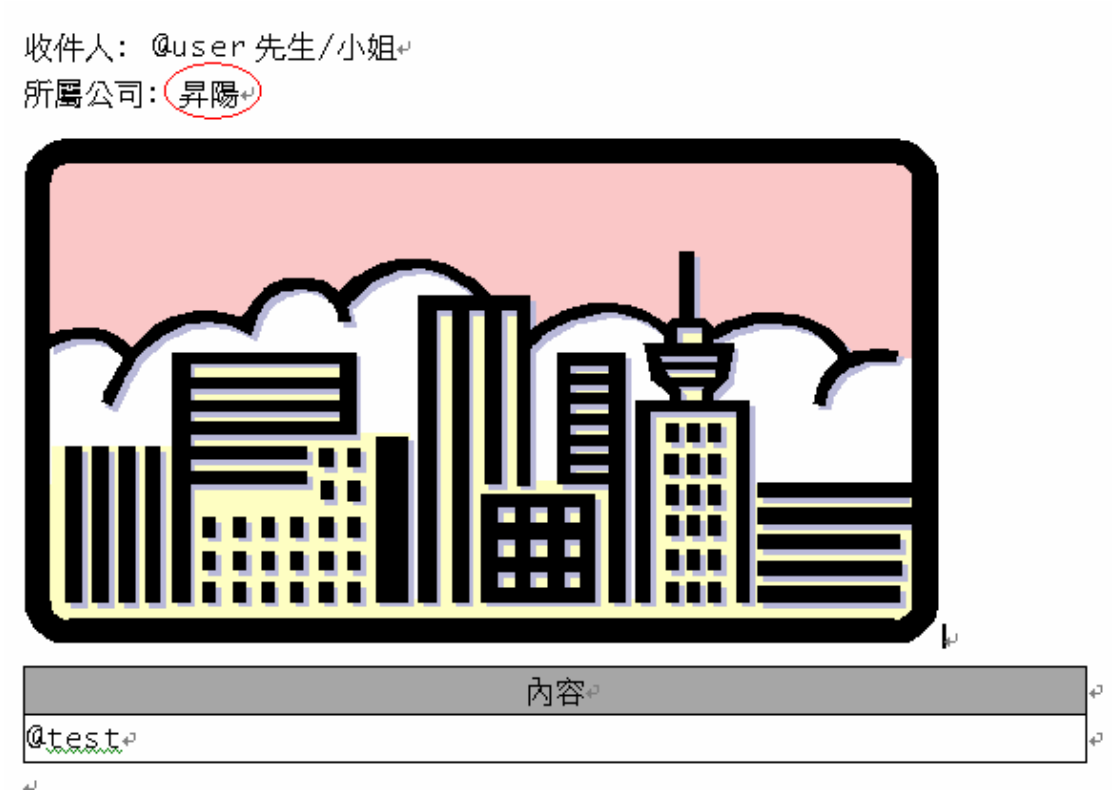
按下確定鈕之後，我們在 Microsoft WORD 裡面所作的每一件事情，都會被錄製成巨集，也就是 VBA 的程式碼。

現在，請按下 Microsoft WORD 選單上的 『編輯 | 取代』

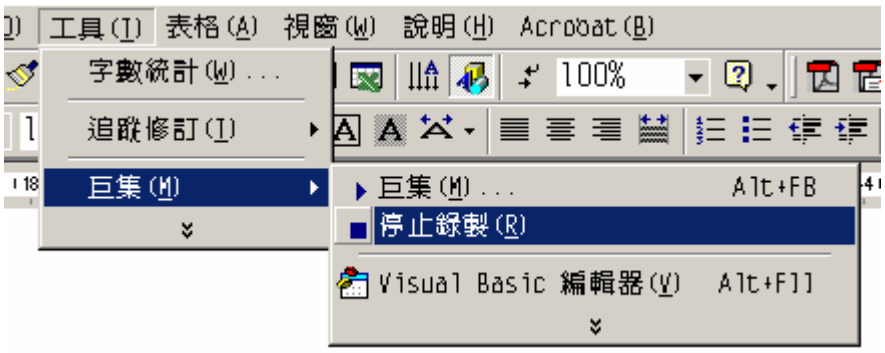
在尋找/取代對話方塊之中填入資料，如下圖所示:

按下全部取代鈕之後，系統會自動完成搜尋/取代的工作，並顯示底下視窗:

同時，我們的樣版檔變成:



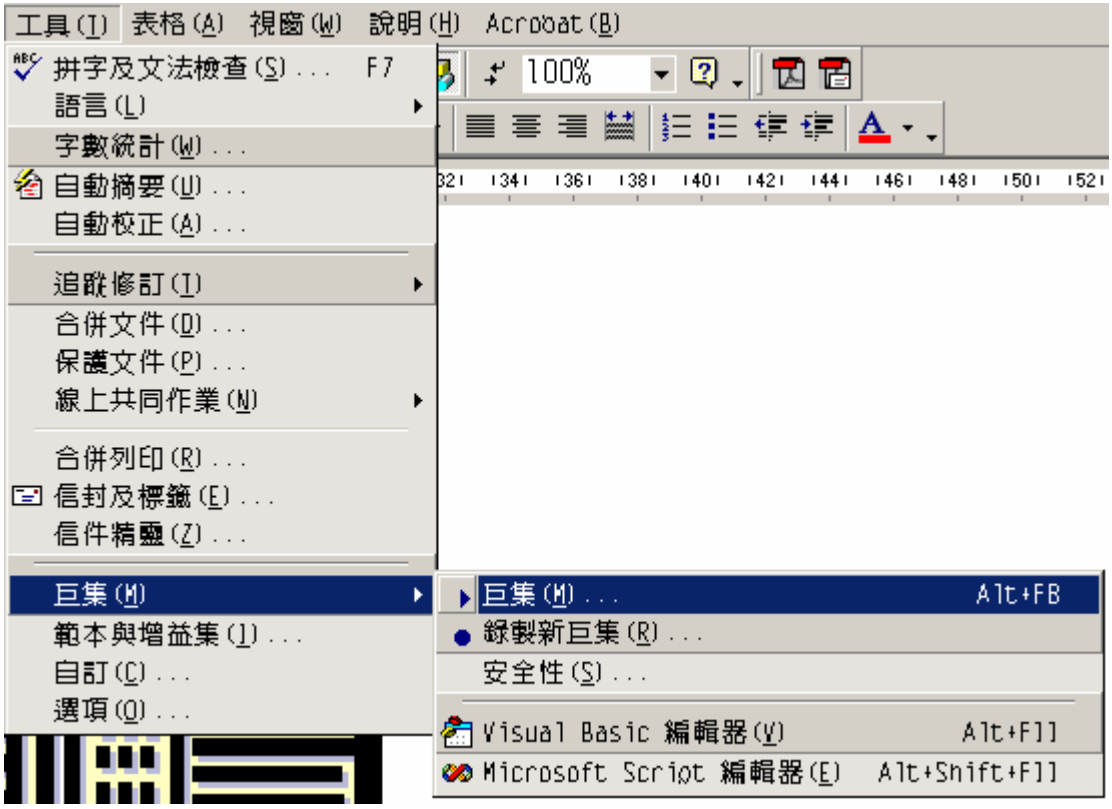
最後，請選擇 Microsoft WORD 選單上的 『工具 | 巨集 | 停止錄製』，如下圖所示:



就可以停止巨集的錄製。

3.觀看巨集

請選擇 Microsoft WORD 選單上的 『工具 | 巨集 | 巨集』，如下圖所示:



在巨集對話方塊之中，選擇名為 test 的巨集，然後按下編輯，如下圖所示:

系統會秀出剛剛我們所作的動作相對應的 VBA 程式碼:

巨集 test 的 VBA 程式碼

```
Sub test()
'
' test 巨集
' 巨集錄製於 2001/12/28，錄製者 moli
'
    Selection.Find.ClearFormatting
    Selection.Find.Replacement.ClearFormatting
    With Selection.Find
        .Text = "@company"
        .Replacement.Text = "昇陽"
        .Forward = True
        .Wrap = wdFindContinue
        .Format = False
        .MatchCase = False
        .MatchWholeWord = False
        .MatchByte = True
        .MatchWildcards = False
        .MatchSoundsLike = False
        .MatchAllWordForms = False
    End With
    Selection.Find.Execute Replace:=wdReplaceAll
End Sub
```

同樣地，如果是一個開啟某個 WORD 文字檔的 VBA 程式碼，經過錄製以後內

容如下:

巨集 test1 的 VBA 程式碼

```
Sub test1()
'
' test1 巨集
' 巨集錄製於 2001/12/28，錄製者 moli
'
    ChangeFileOpenDirectory "E:\Java2Word\"
    Documents.Open FileName:="test.doc", ConfirmConversions:=False, ReadOnly:= _
        False, AddToRecentFiles:=False, PasswordDocument:="", PasswordTemplate:= _
        "", Revert:=False, WritePasswordDocument:="", WritePasswordTemplate:="", _
        Format:=wdOpenFormatAuto
End Sub
```

有了上述兩組巨集指令之後，我們就可以用 C++撰寫完整的套表列印功能模組了。

功能模組建構程序六: 完成套表列印功能模組

好了，我們已經完成了所有的前置工作，我們可以動手撰寫套表列印功能模組的程式碼。請啟動您的 C++ Builder，並開啟 Transfer.bpr 這個專案檔。我們將 Transfer.cpp 的內容依以下步驟修改:

步驟一: 加入引入檔

請先加入幾個使用 COM 技術(OLE Automation 的基礎架構)時必須用到的引入檔，他們分別是:OleServer.hpp、utilcls.h、comobj.hpp。另外，要使用 Microsoft WORD 2000 所提供的軟體元件，還必須要額外引入 vcl\word_2k.h(在安裝 C++Builder 的時候，安裝程式會根據您電腦上的 Office 版本產生對應的引入檔，所以如果您編譯 Transfer.cpp 時，編譯器說找不到此檔，那麼請重新檢視您電腦上 C++Builder 的安裝程序，筆者已測試過 Professional 版與 Enterprise 版，一切正常)。修改後的 Transfer.cpp 如下:

Transfer.cpp

```
#include <vcl.h>
#include <OleServer.hpp>
#include <windows.h>
#include "com_sun_edu_JavatoWord.h"
#include <utilcls.h>
#include <comobj.hpp>
#include <iostream.h>
#include <vcl\word_2k.h>
```

```
#pragma hdrstop

#pragma argsused
int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason, void* lpReserved)
{
        return 1;
}

JNIEXPORT void JNICALL Java_com_sun_edu_JavatoWord_NativeToWord
  (JNIEnv *env, jobject obj, jobjectArray context, jint count)
{
        ...略...
}
```

步驟二: 加入使用 COM 技術時必須叫用的 API

根據規定，使用 COM 技術之前，必須呼叫 CoInitialize(NULL)，而離開之前，需要叫用 CoUninitialize()。因此程式碼需要修改成底下的樣子:

Transfer.cpp

```
...略...

JNIEXPORT void JNICALL Java_com_sun_edu_JavatoWord_NativeToWord
  (JNIEnv *env, jobject obj, jobjectArray context, jint count)
{
        //讀入樣板檔
        TOpenDialog* templatefile = new TOpenDialog(NULL) ;
        templatefile->Title = "請選擇樣板檔" ;
        templatefile->Filter = "Word files (*.doc)|*.doc" ;
        AnsiString filename ;
        if(templatefile->Execute())
        {
                filename = templatefile->FileName ;
        }
        //檢查樣板檔是否存在
        if(!FileExists(filename))
        {
                ShowMessage("您所輸入的檔案並不存在,請重新輸入") ;
```

```
                delete templatefile ;
                return ;
        }
        delete templatefile ;

        CoInitialize(NULL) ;
        try
        {

        }catch(Exception& e)
        {
                ShowMessage(e.Message) ;
                CoUninitialize() ;
                return ;
        }
        CoUninitialize() ;
}
```

步驟三: 使用 WORD 來開啟 WORD 文字檔

根據我們所取得的 test1 巨集內容，我們可以得知要如何使用 WORD 來開啟 WORD 文字檔:

```
Sub test1()
'
' test1 巨集
' 巨集錄製於 2001/12/28，錄製者 moli
'
    ChangeFileOpenDirectory "E:\Java2Word\"
    Documents.Open FileName:="test.doc", ConfirmConversions:=False, ReadOnly:= _
        False, AddToRecentFiles:=False, PasswordDocument:="", PasswordTemplate:= _
        "", Revert:=False, WritePasswordDocument:="", WritePasswordTemplate:="", _
        Format:=wdOpenFormatAuto
End Sub
```

從巨集內容可以得知，需要使用 Documents 物件的 Open 功能來開啟，因此 Transfer.cpp 需要修改如下:

Transfer.cpp

```
...略...

JNIEXPORT void JNICALL Java_com_sun_edu_JavatoWord_NativeToWord
  (JNIEnv *env, jobject obj, jobjectArray context, jint count)
{
        //讀入樣板檔
        TOpenDialog* templatefile = new TOpenDialog(NULL) ;
```

```
        templatefile->Title = "請選擇樣板檔" ;
        templatefile->Filter = "Word files (*.doc)|*.doc" ;
        AnsiString filename ;
        if(templatefile->Execute())
        {
                filename = templatefile->FileName ;
        }
        //檢查樣板檔是否存在
        if(!FileExists(filename))
        {
                ShowMessage("您所輸入的檔案並不存在,請重新輸入") ;
                delete templatefile ;
                return ;
        }
        delete templatefile ;

        CoInitialize(NULL) ;
        try
        {
                Variant Wordobj ;
                Variant Docsobj ;
                //產生 Word 物件
               Wordobj = CreateOleObject("Word.Application") ;
               cout << "建立 Word 物件成功" <<endl   ;
               Wordobj.OlePropertySet("Visible",true) ;

               //開啟 user 所指定的樣版檔
               Docsobj = Wordobj.OlePropertyGet("Documents") ;
               Docsobj.OleFunction("Open",filename) ;
        }catch(Exception& e)
        {
                ShowMessage(e.Message) ;
                CoUninitialize() ;
                return ;
        }
        CoUninitialize() ;

}
```

修改完成後，請使用選單上的 Project | Make Transfer 重新建立動態聯結函式庫(.dll 檔)。然後到 DOS 視窗，執行指令:

java –cp . –Djava.library.path=.\Transfer test

如果執行正常，您將發現您在選擇樣版對話方塊之中所指定的檔名，已經由 WORD 幫我們開啟。

在我們的程式之中，首先建立了一個名為 Word.Application 的物件，這個物件代表的是整個 Microsoft WORD，所以要在 C/C++之中利用 OLE Automation 來操控 WORD，第一個步驟一定要先產生此物件。同理，如果您要操控 Microsoft EXCEL，您也必須先產生 Excel.Application 物件。

有了 Word.Application 物件之後，由於預設的情況下 WORD 是隱藏的，我們必須讓他顯示在螢幕上，所以我們利用 OlePropertySet 函式，將 Application 物件(底下我們直接以 Application 物件來稱呼 Word.Application)的 Visible 屬性設為 true。有關 Visible 屬性，WORD 內附的 Visual Basic 編輯器輔助說明如下:

Visible 屬性

請參閱　範例　適用於

Application、**Window**、**Task** 或 **Border** 物件：如果看得到指定物件，則為 **True**。可讀寫的 **Boolean** 資料型態。

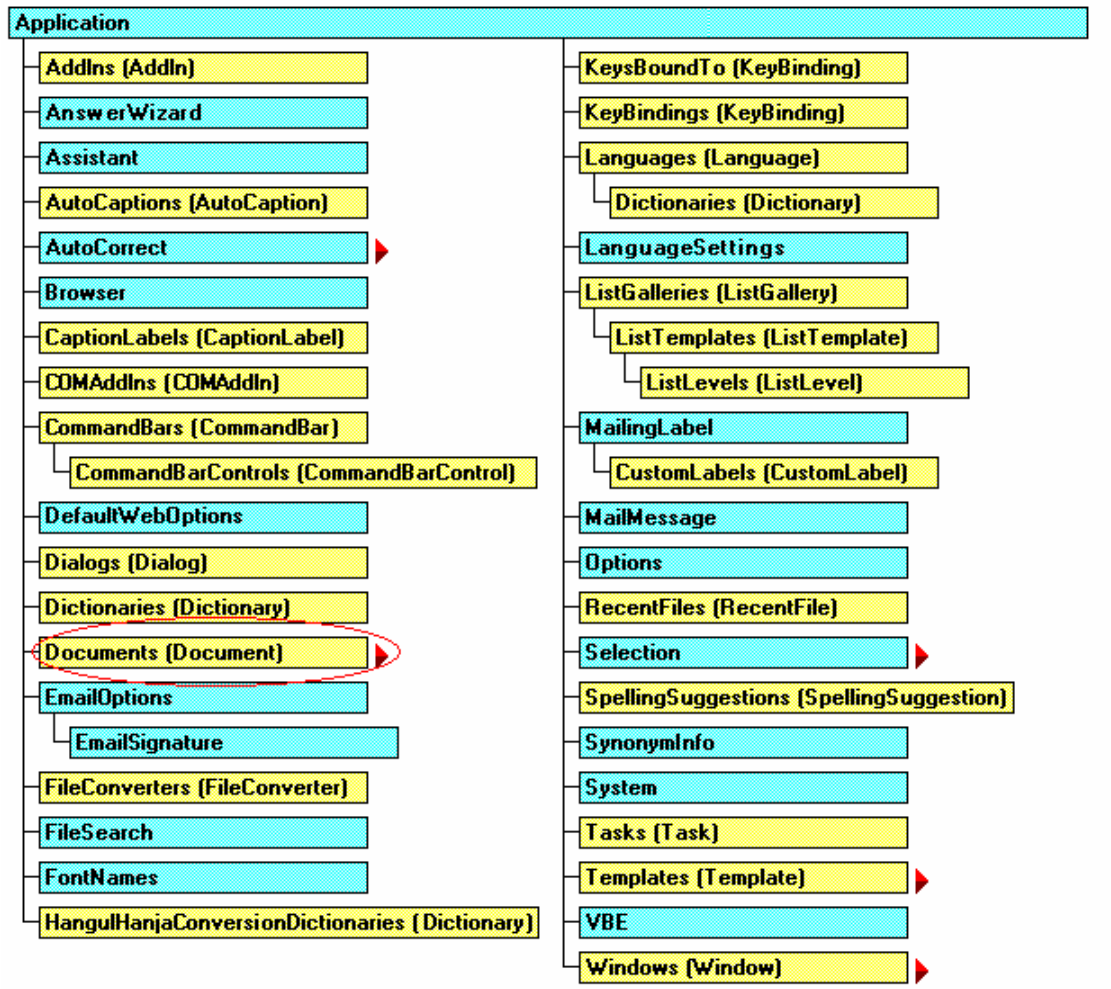
FillFormat、**LineFormat**、**ShadowFormat**、**Shape**、**ShapeRange** 或 **ThreeDFormat** 物件：如果看得到指定物件或該物件的格式設定，則為 **True**。可讀寫的 **Long** 資料型態。

註解

對於任何物件，如果 **Visible** 為 **False**，有些方法及屬性可能會無效。

(上圖取自於 Microsoft WORD 內附 Visual Basic 編輯器的輔助說明，版權屬於 Microsoft 所有)

接下來，我們從 test1 巨集之中得知，我們必須使用 Documents 物件的 Open 方法來開啟檔案。於是，我們查詢了 Microsoft WORD 中的物件關係圖，發現 Documents 物件屬於 Application 物件的直屬物件，如下圖所示



(上圖取自於 Microsoft WORD 內附 Visual Basic 編輯器的輔助說明，版權屬於 Microsoft 所有)

所以我們可以直接利用 Application 物件來取得 Documents 物件，然後叫用其 Open 方法，並給予其欲開啟的檔案路徑和名稱即可。有關 Open 方法的描述，請參閱 Microsoft WORD 內附 Visual Basic 編輯器的輔助說明。

步驟四: 使用取代功能

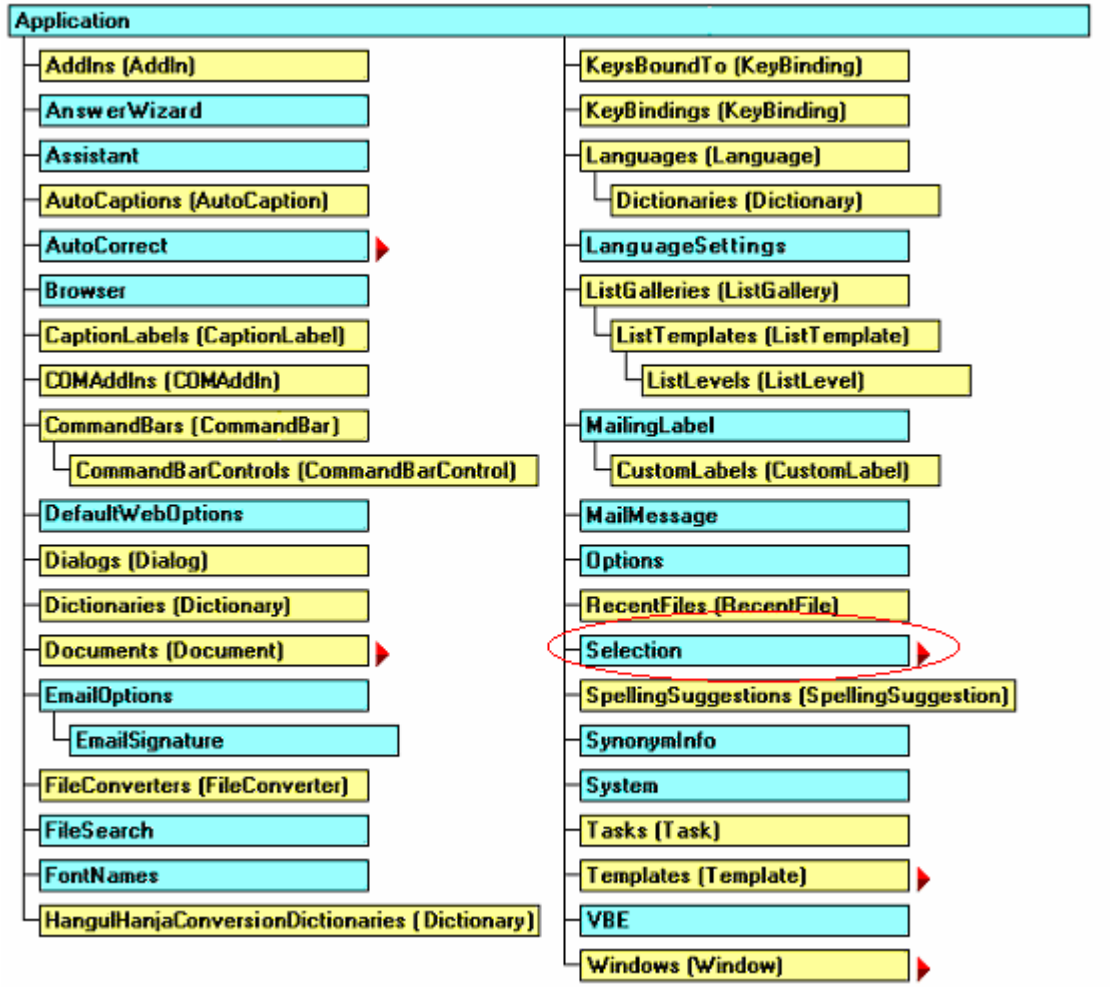
根據我們所取得的 test 巨集內容，我們可以得知要如何使用 WORD 內建的文字取代功能:

```
Sub test()
'
' test 巨集
' 巨集錄製於 2001/12/28，錄製者 moli
'
    Selection.Find.ClearFormatting
    Selection.Find.Replacement.ClearFormatting
    With Selection.Find
        .Text = "@company"
        .Replacement.Text = "昇陽"
        .Forward = True
        .Wrap = wdFindContinue
        .Format = False
        .MatchCase = False
        .MatchWholeWord = False
        .MatchByte = True
        .MatchWildcards = False
        .MatchSoundsLike = False
        .MatchAllWordForms = False
    End With
    Selection.Find.Execute Replace:=wdReplaceAll
End Sub
```

函式名稱

參數

根據物件關係圖，我們可以知道 Selection 物件屬於 Application 物件的直屬物件，如下圖所示:



(上圖取自於 Microsoft WORD 內附 Visual Basic 編輯器的輔助說明，版權屬於 Microsoft 所有)

如果您使用 Microsoft WORD 內附 Visual Basic 編輯器的輔助說明，只要按下上圖 Selection 右邊的紅色箭頭，就可以依序找到與 Selection 物件有相依關係

的 Find 物件您可以查詢到屬於 Find 物件的 ClearFormatting 方法與 Execute 方法。再按下 Find 物件右邊的紅色箭頭，就可以依序找到與 Find 物件有相依關係的 Replacement 物件，最後就可以找到其 ClearFormatting 方法。因此，最後的程式碼如下:

Transfer.cpp

```
...略...

JNIEXPORT void JNICALL Java_com_sun_edu_JavatoWord_NativeToWord
  (JNIEnv *env, jobject obj, jobjectArray context, jint count)
{
        //讀入樣板檔
        TOpenDialog* templatefile = new TOpenDialog(NULL) ;
        templatefile->Title = "請選擇樣板檔" ;
        templatefile->Filter = "Word files (*.doc)|*.doc" ;
        AnsiString filename ;
        if(templatefile->Execute())
        {
                filename = templatefile->FileName ;
        }
        //檢查樣板檔是否存在
        if(!FileExists(filename))
        {
                ShowMessage("您所輸入的檔案並不存在,請重新輸入") ;
                delete templatefile ;
                return ;
        }
        delete templatefile ;
        CoInitialize(NULL) ;
        try
        {
                Variant Wordobj ;
                Variant Docsobj ;
                //產生 Word 物件
               Wordobj = CreateOleObject("Word.Application") ;
               cout << "建立 Word 物件成功" <<endl  ;
               Wordobj.OlePropertySet("Visible",true) ;
```

```
        //開啟 user 所指定的樣版檔
        Docsobj = Wordobj.OlePropertyGet("Documents") ;
        Docsobj.OleFunction("Open",filename) ;

        Variant ActDoc ;
        Variant Selection ;
        Variant Find ;
        Variant Replacement ;
        //取得目前作用的 document
        ActDoc = Wordobj.OlePropertyGet("ActiveDocument") ;
        //取得 Selection.Find.Replacement 物件
        Selection = Wordobj.OlePropertyGet("Selection") ;
        Find = Selection.OlePropertyGet("Find") ;
        Replacement = Find.OlePropertyGet("Replacement") ;
        //執行所有來自 Java 程式中,所請求的取代動作
        for(int i = 0 ; i < count ; i++)
        {
                //取出 context 的內容
                jstring jsdata =
                (jstring)(*env).GetObjectArrayElement(context,i) ;
                wchar_t* csdata =
                (wchar_t*)(*env).GetStringChars(jsdata,NULL) ;
                int size = (*env).GetStringLength(jsdata) ;
                WideString wtmp = WideString(csdata,size) ;
                int start = wtmp.Pos(":") ;
                WideString text = wtmp.SubString(1,start-1) ;
                text = "@" + text ;
                WideString rtext =
                    wtmp.SubString(start+1,wtmp.Length() - start) ;

                //開始取代動作
                Find.OleFunction("ClearFormatting") ;
                Replacement.OleFunction("ClearFormatting") ;
                Find.OleFunction("Execute",text,true,false,false,
                false,false,true,wdFindContinue,false,rtext,wdReplaceAll) ;
         }
}catch(Exception& e)
{
```

```
                ShowMessage(e.Message) ;
                CoUninitialize() ;
                return ;
        }
        CoUninitialize() ;
}
```

修改完成後，請使用選單上的 Project | Make Transfer 重新建立動態聯結函式庫(.dll 檔)。然後到 DOS 視窗，執行指令:

java –cp . –Djava.library.path=.\Transfer test

如果執行正常，您將發現此模組已經完成了我們所期望的功能。

總結

作為一個佔有率最高的的辦公室軟體套件，Microsoft Office 在功能上的確有其獨到的地方。尤其是，Microsoft Office 裡頭有很多非常優秀、且可以重複使用的軟體元件，幾乎都被我們忽略了。其實，多多善用 Microsoft Office 的軟體元件所提供的強大功能，可以減少程式設計師沉重的負擔，例如 Excel 的強大計算功能與圖表功能就是一個很好的例子，當然，使用了 JNI 之後，您的 Java 程式就失去了跨平台的特性。這是諸位讀者必須要取捨的地方。

希望各位閱讀過本章之後，將來有必要時，可以善加利用 Microsoft Office 既有的軟體元件(其實不只只有 Microsoft Office，其他如 Microsoft Visio、Microsoft Internet Explorer、還有 AutoCAD，幾乎只要可以用 VBA 來操控的軟體，都有此功能)來加強自己開發的軟體。

Java Native Interface 參考資源

利用 Java 配合 BCB 4.0 製作 CPU 特徵偵測器
作者 ： 王森
(網路版本: http://www.javatwo.net/experts/moli/publish/article/TC/BCB_CPU_Detector.htm)
遊戲設計大師　NO.8　1999 年 11~12 月號

Essential Jni : Java Native Interface
作者 : Rob Gordon, Robert Gordon, Alan McClellan

http://www.amazon.com/exec/obidos/ASIN/0136798950/o/qid=990362161/sr=8-1/ref=aps_sr_b_1_1/002-7278315-6856028

The Java Native Interface : Programmer's Guide and Specification

作者 : Sheng Liang

http://www.amazon.com/exec/obidos/ASIN/0201325772/o/qid=990362161/sr=8-2/ref=aps_sr_b_1_2/002-7278315-6856028